Propeller-GCC Libraries

Generated by Doxygen 1.7.1

Tue Jun 5 2012 15:39:11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 __driver Struct Reference

Generic and customizable driver struct for stdio devices.

`#include <driver.h>`

Public Attributes

- const char * prefix
- int(* fopen)(FILE *fp, const char *name, const char *mode)
- int(* fclose)(FILE *fp)
- int(* read)(FILE *fp, unsigned char *buf, int size)
- int(* write)(FILE *fp, unsigned char *buf, int size)
- int(* seek)(FILE *fp, long offset, int whence)
- int(* remove)(const char *name)
- int(* getbyte)(FILE *fp)
- int(* putbyte)(int c, FILE *fp)

3.1.1 Detailed Description

Generic and customizable driver struct for stdio devices. The purpose is to allow replacing the stdio functions. Any device can be attached to stdio functions with this struct.

Typically the __driver array is defined by users for setting up stdin, stdout, stderr FILE drivers, SD card FILE drivers, and others.

The _Driver list is a list of all drivers we can use in the program. The default _InitIO function opens stdin, stdout, and stderr based on the first driver in the list (typically the serial driver). The serial driver could be replaced by a TV/Keyboard driver.

When defined by the user, the array of structs may look like this:

```
extern _Driver _SimpleSerialDriver;
extern _Driver _FileDriver;

_Driver *_driverlist[] = {
  &_SimpleSerialDriver,
```

```
  &_FileDriver,
  NULL
};
```

The NULL driver ends the _Driver list.

The device driver interface is the __driver struct. By defining the struct in the device driver, one connects driver functions to the _Driver.

In a TV output device driver for example, we need to define fopen() for the _InitIO() routine, fwrite(), and fclose().

```
_Driver TvDriver = {
  TvPrefix,     // TvPrefix is the device driver name "TV"
  Tv_fopen,     // fopen starts the TV COG (term for Propeller core).
  Tv_fclose,    // fclose stops the TV COG if necessary, etc....
  _null_read,   // use _null_read instead of defining Tv_fread
  Tv_write,     // fwrite is used to send characters to the TV
  NULL,         // seek; not applicable
  NULL,         // remove; not applicable
};
```

Of course it is not necessary to use a stdio method for TV output. There are some cases where the stdio infrastructure is not necessary. The standard stdio library is relatively big, but as small as possible.

The __driver mechanism gives the program flexibility in a standard way. The types of drivers are limited only by the programmer's imagination.

Some VGA demo programs have been written entirely in C.

Definition at line 86 of file driver.h.

3.1.2 Member Data Documentation

3.1.2.1 int(* __driver::fclose)(FILE *fp)

Prototype for closing files to be filled in by user's driver.

Parameters

`[in,out]` ***fp*** FILE pointer set by previous fopen() call.

Returns

0 on success, nonzero on failure

Definition at line 115 of file driver.h.

3.1.2.2 int(* __driver::fopen)(FILE *fp, const char *name, const char *mode)

Prototype for opening files.

A function for the user's device must be provided by the user's driver.

Parameters

[out] ***fp*** This is the file pointer for the driver.

[in] ***name*** This is the device name string.

[in] ***mode*** This is the device open mode string.

Returns

0 on success, nonzero on failure

Definition at line 108 of file driver.h.

3.1.2.3 int(* __driver::getbyte)(FILE *fp)

Optional prototype for single character input.

Function getbyte is needed for reading the generic terminal driver.

Parameters

[in] ***fp*** The file pointer.

Returns

character read by the function.

Definition at line 162 of file driver.h.

3.1.2.4 const char* __driver::prefix

The file "device" prefix for fopen.

This string allows users to name their devices an pass startup information to the driver if nessesary.

Some library device names: "SSER", "FDS", "TV", etc....

The default Propeller-GCC stdio console device name is "SSER".

Definition at line 98 of file driver.h.

3.1.2.5 int(* __driver::putbyte)(int c, FILE *fp)

Optional prototype for single character output.

Function putbyte is needed for writing to the generic terminal driver.

Parameters

c The character to write.

[in] ***fp*** The file pointer.

Returns

character read by the function.

Definition at line 171 of file driver.h.

3.1.2.6 int(* __driver::read)(FILE *fp, unsigned char *buf, int size)

Prototype for reading multi byte I/O.

A function for the user's device must be provided by the user's driver.

Parameters

`[in,out]` ***fp*** FILE pointer set by previous fopen() call.

`[out]` ***buf*** A char buffer of at least size length where data is put after read.

`[in]` ***size*** The size of the buf parameter.

Returns

The number of bytes read. If an error occurs, or the end-of-file is reached, the return value is a short object count (or zero).

Definition at line 126 of file driver.h.

3.1.2.7 int(* __driver::remove)(const char *name)

Prototype for removing a file or directory from the file system.

Parameters

name The name of the file to remove.

Returns

Zero on success, or -1 on error with errno set to indicate error.

Definition at line 154 of file driver.h.

3.1.2.8 int(* __driver::seek)(FILE *fp, long offset, int whence)

Prototype for seek to a position in the file.

A function for the user's device must be provided by the user's driver.

Parameters

`[in,out]` ***fp*** FILE pointer set by previous fopen() call.

`[in]` ***offset*** The offset to add to the file position specified by whence.

`[in]` ***whence*** The start position specifier: SEEK_SET, SEEK_CUR, or SEEK_END.

Returns

Zero on success, or -1 on error with errno set to indicate error.

Definition at line 147 of file driver.h.

3.1.2.9 int(* __driver::write)(FILE *fp, unsigned char *buf, int size)

Prototype for writing multi byte I/O.

A function for the user's device must be provided by the user's driver.

Parameters

`[in,out]` ***fp*** FILE pointer set by previous fopen() call.

`[in]` ***buf*** A char buffer of at least size length where data is put befoe write.

size The size of the buf to write.

Returns

The number of bytes read. If an error occurs, or the end-of-file is reached, the return value is a short object count (or zero).

Definition at line 137 of file driver.h.

The documentation for this struct was generated from the following file:

- include/sys/driver.h

3.2 dirent Struct Reference

Defines the dirent.h dirent struct.

`#include <dirent.h>`

3.2.1 Detailed Description

Defines the dirent.h dirent struct. dirent fields:

- uint8_t name[11]; // filename
- uint8_t attr; // attributes
- uint8_t reserved; // reserved, must be 0
- uint8_t crttimetenth; // create time, 10ths of a second (0-199)
- uint8_t crttime_0; // creation time byte 0
- uint8_t crttime_1; // creation time byte 1
- uint8_t crtdate_0; // creation date byte 0
- uint8_t crtdate_1; // creation date byte 1
- uint8_t lstaccdate_l; // last access date byte 0
- uint8_t lstaccdate_h; // last access date byte 1
- uint8_t startcluster_2; // first cluster, byte 2 (FAT32)
- uint8_t startcluster_3; // first cluster, byte 3 (FAT32)

- uint8_t wrttime_0; // last write time byte 0
- uint8_t wrttime_1; // last write time byte 1
- uint8_t wrtdate_0; // last write date byte 0
- uint8_t wrtdate_1; // last write date byte 1
- uint8_t startcluster_0; // first cluster, byte 0
- uint8_t startcluster_1; // first cluster, byte 1
- uint8_t filesize_0; // file size, byte 0
- uint8_t filesize_1; // file size, byte 1
- uint8_t filesize_2; // file size, byte 2
- uint8_t filesize_3; // file size, byte 3
- char d_name[13]; // filename in file.ext format

Definition at line 65 of file dirent.h.

The documentation for this struct was generated from the following file:

- include/dirent.h

3.3 DIRstruct Struct Reference

Defines the dirent.h DIR struct.

`#include <dirent.h>`

3.3.1 Detailed Description

Defines the dirent.h DIR struct. DIR fields:

- uint32_t currentcluster; // current cluster in dir
- uint8_t currentsector; // current sector in cluster
- uint8_t currententry; // current dir entry in sector
- uint8_t *scratch; // ptr to user-supplied sector buffer
- uint8_t flags; // internal DOSFS flags

Definition at line 29 of file dirent.h.

The documentation for this struct was generated from the following file:

- include/dirent.h

3.4 lconv Struct Reference

lconv contains members related to the formatting of numeric values.

`#include <locale.h>`

Public Attributes

- char * decimal_point

  The decimal-point character used to format non-monetary quantities.

- char * thousands_sep

  The character used to separate groups of digits in non-monetary quantities.

- char * grouping

  A string that indicates the size of each group of digits in non-monetary quantities.

- char * mon_decimal_point

  The decimal-point used to format monetary quantities.

- char * mon_thousands_sep

  The separator for groups of digits before the decimal-point in monetary quantities.

- char * mon_grouping

  A string that indicates the size of each group of digits in monetary quantities.

- char * positive_sign

  A string that indicates a non-negative formatted value in monetary quantities.

- char * negative_sign

  A string that indicates a negative formatted value in monetary quantities.

- char * currency_symbol

  The currency symbol applicable in the current locale.

- char frac_digits

  The number of fractional digits to be displayed in a locally formatted monetary quantity.

- char p_cs_precedes

  Set to 1 or 0 if currency_symbol preceeds or succeeds respectively the non-negative value of monetary quantity.

- char n_cs_precedes

  Set to 1 or 0 if currency_symbol preceeds or succeeds respectively the negative value of monetary quantity.

- char p_sep_by_space

  Set to a value indicating the separation of the currency_symbol, the sign, and the value for non-negative monetary quantity.

- char n_sep_by_space

Set to a value indicating the separation of the currency_symbol, the sign, and the value for negative monetary quantity.

- char p_sign_posn

  Set to a value indicating the position of the positive_sign for non-negative monetary quantity.

- char n_sign_posn

  Set to a value indicating the position of the positive_sign for negative monetary quantity.

- char * int_curr_symbol

  The international currency symbol applicable to the current locale.

- char int_frac_digits

  The number of fractional digits to be displayed in a monetary quantity.

- char int_p_cs_precedes

  Set to 1 or 0 if the int_currency_symbol precedes or succeeds respectively the value for a non-negative internationally formatted monetary quantity.

- char int_n_cs_precedes

  Set to 1 or 0 if the int_currency_symbol precedes or succeeds respectively the value for a negative internationally formatted monetary quantity.

- char int_p_sep_by_space

  Set to a value indicating the separation of the int_currency_symbol the sign string, and the value for non-negative international monetary formatted quantities.

- char int_n_sep_by_space

  Set to a value indicating the separation of the int_currency_symbol the sign string, and the value for negative international monetary formatted quantities.

- char int_p_sign_posn

  Set to a value indicating the position of the positive_sign for non-negative international monetary formatted quantity.

- char int_n_sign_posn

  Set to a value indicating the position of the positive_sign for negative international monetary formatted quantity.

3.4.1 Detailed Description

lconv contains members related to the formatting of numeric values.

Definition at line 59 of file locale.h.

3.4.2 Member Data Documentation

3.4.2.1 char* lconv::int_curr_symbol

The international currency symbol applicable to the current locale.

The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217:1995. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

Definition at line 149 of file locale.h.

The documentation for this struct was generated from the following file:

- include/locale.h

3.5 pthread_attr_t Struct Reference

The pthread_attr_t struct TODO.

`#include <pthread.h>`

Public Attributes

- size_t stksiz
  stack size
- void * stack
  pointer to base of stack, NULL to allocate one
- unsigned int flags
  flags to start with

3.5.1 Detailed Description

The pthread_attr_t struct TODO.

Definition at line 63 of file pthread.h.

The documentation for this struct was generated from the following file:

- include/pthread.h

3.6 pthread_mutex_t Struct Reference

The pthread_mutex_t TODO.

`#include <pthread.h>`

3.6.1 Detailed Description

The pthread_mutex_t TODO.

Definition at line 76 of file pthread.h.

The documentation for this struct was generated from the following file:

- include/pthread.h

Chapter 4

File Documentation

4.1 include/assert.h File Reference

Provides assert interface used for internal error detection.

`#include <compiler.h>`

Defines

- #define assert(cond) ((cond) ? 0 : (__eprintf(#cond,(long)(__LINE__), __FILE__), abort(), 0))

  Abort the program if the assertion is false.

4.1.1 Detailed Description

Provides assert interface used for internal error detection.

Definition in file assert.h.

4.1.2 Define Documentation

4.1.2.1 #define assert(*cond*) ((cond) ? 0 : (__eprintf(#cond,(long)(__LINE__), __FILE__), abort(), 0))

Abort the program if the assertion is false.

If macro NDEBUG is defined at the moment <assert.h> was last included the assert() macro genenrates no code and does nothing.

Else assert() prints an error message to stderr and kills the program by calling abort() if cond is false.

Parameters

cond The condition tested

Definition at line 34 of file assert.h.

4.2 include/cog.h File Reference

Inludes common API definitions for COG programming.

Defines

- #define _COGMEM __attribute__((cogmem))

  Can be used in per-variable declarations to tell compiler that a variable should go in COG memory.

- #define _NATIVE __attribute__((native))

  Can be used in per-function declarations to tell compiler that function will use cog "call/ret" calling (non-recursive).

- #define _NAKED __attribute__((naked))

  Can be used in per-function declarations to tell compiler that function will not have an epilogue or prologue: these should never return.

- #define _CLKFREQ _clkfreq

  This is an alias for the 32 bit clock frequency which is kept in address 0.

- #define _CLKMODE _clkmode

  This is an alias for the 8 bit clock mode which is kept in address 4.

Variables

- unsigned int _clkfreq
- unsigned char _clkmode

4.2.1 Detailed Description

Inludes common API definitions for COG programming. Each COG includes 16 32 bit special purpose registers. The purposes of the registers are to provide control over user I/O input and output. Some registers like INB, OUTB, DIRB are not used with P8X32A.

The state of each physical input pin is available to any COG via INA. Output pin values are the bitwise "wire OR" of all the COGs at the physical output pins when the DIRA bits are set high (1).

OUTA bits control the state of the physical output pins. If one COG sets a pin to output high (1), and another COG sets the same pin to output low (0), the high (1) will be set.

The per COG special purpose register summary:

```
Register Description

PAR      Parameter register is used for sharing HUB RAM address info with the COG.
CNT      The system clock count
INA      Use to read the pins when corresponding DIRA bits are 0.
INB      Unused in P8X32A
OUTA     Use to set pin states when corresponding DIRA bits are 1.
OUTB     Unused in P8X32A
DIRA     Use to set pins to input (0) or output (1).
DIRB     Unused in P8X32A
```

```
CTRA     Counter A control register.
CTRB     Counter B control register.
FRQA     Counter A frequency register.
FRQB     Counter B frequency register.
PHSA     Counter A phase accumulation register.
PHSB     Counter B phase accumulation register.
VCFG     Video Configuration register can be used for other special output.
VSCL     Video Scale register for setting pixel and frame clocks.
```

Definition in file cog.h.

4.2.2 Define Documentation

4.2.2.1 #define _CLKMODE _clkmode

This is an alias for the 8 bit clock mode which is kept in address 4.

This is not automatically updated by the clkset macro.

Definition at line 93 of file cog.h.

4.2.3 Variable Documentation

4.2.3.1 unsigned int _clkfreq

32 bit system startup clock frequency variable

4.2.3.2 unsigned char _clkmode

System startup clock mode

4.3 include/cogload.h File Reference

Provides Propeller specific functions to load COGs from EEPROM.

```
#include <stdint.h>
#include <i2c.h>
```

Functions

- int cognewFromBootEeprom (void *code, size_t codeSize, void *param)

  Start a new Propeller PASM COG from a COG image in EEPROM.

4.3.1 Detailed Description

Provides Propeller specific functions to load COGs from EEPROM. Copyright (c) 2011-2012 by Parallax, Inc. MIT Licensed

Definition in file cogload.h.

4.3.2 Function Documentation

4.3.2.1 int cognewFromBootEeprom (void * *code*, size_t *codeSize*, void * *param*)

Start a new Propeller PASM COG from a COG image in EEPROM.

This is use to start a COG with code compiled as PASM, AS, or COG-C when the COG image is in EEPROM. PASM can be any Spin/PASM code that is self-contained. That is, all data for initialization and mailbox use are passed via the par parameter. Changing PASM variables from SPIN code will not work with this method.

GAS and COG-C programs have similar restrictions. COG-C programs should not use any stack or variables in HUB memory that are not accessed via PAR mailbox or pointers.

Parameters

code Address of PASM to load

size of code in bytes

param Value of par parameter usually an address

Returns

COG ID provided by the builtin function or -1 on failure.

4.4 include/compiler.h File Reference

Defines features of the compiler being used.

4.4.1 Detailed Description

Defines features of the compiler being used. This file defines various features of the compiler being used such as:

- _INT_SIZE 2 for 16 bit, 4 for 32 bit
- _LONG_SIZE 4 for 32 bit, 8 for 64 bit
- _WCHAR_SIZE sizeof(wchar_t)
- _CHAR_IS_UNSIGNED 1 or 0, depending
- _NORETURN: attribute indicating a function does not return
- _CONST: indicates that the function does not change or examine memory

Definition in file compiler.h.

4.5 include/complex.h File Reference

Provides complex math API declarations.

Functions

- double complex cacos (double complex z)

  Complex inverse cosine function.

- float complex cacosf (float complex z)
- __ldouble_t complex cacosl (__ldouble_t complex z)
- double complex casin (double complex z)

  Complex inverse sine function.

- float complex casinf (float complex z)
- __ldouble_t complex casinl (__ldouble_t complex z)
- double complex catan (double complex z)

  Complex inverse tangent function.

- float complex catanf (float complex z)
- __ldouble_t complex catanl (__ldouble_t complex z)
- double complex ccos (double complex z)

  Complex cosine function.

- float complex ccosf (float complex z)
- __ldouble_t complex ccosl (__ldouble_t complex z)
- double complex csin (double complex z)

  Complex sine function.

- float complex csinf (float complex z)
- __ldouble_t complex csinl (__ldouble_t complex z)
- double complex ctan (double complex z)

  Complex tangent function.

- float complex ctanf (float complex z)
- __ldouble_t complex ctanl (__ldouble_t complex z)
- double complex cacosh (double complex z)

  complex inverse hyperbolic cosine function.

- float complex cacoshf (float complex z)
- __ldouble_t complex cacoshl (__ldouble_t complex z)
- double complex casinh (double complex z)

  Complex inverse hyperbolic sine function.

- float complex casinhf (float complex z)
- __ldouble_t complex casinhl (__ldouble_t complex z)
- double complex catanh (double complex z)

  Complex inverse hyperbolic tangent function.

- float complex catanhf (float complex z)
- __ldouble_t complex catanhl (__ldouble_t complex z)
- double complex ccosh (double complex z)

  Complex hyperbolic cosine function.

- float complex ccoshf (float complex z)
- __ldouble_t complex ccoshl (__ldouble_t complex z)
- double complex csinh (double complex z)

  Complex hyperbolic sine function.

- float complex csinhf (float complex z)
- __ldouble_t complex csinhl (__ldouble_t complex z)
- double complex ctanh (double complex z)

  Complex hyperbolic tangent function.

- float complex ctanhf (float complex z)
- __ldouble_t complex ctanhl (__ldouble_t complex z)
- double complex cexp (double complex z)

  Complex expotential function returns e raised to z power.

- float complex cexpf (float complex z)
- __ldouble_t complex cexpl (__ldouble_t complex z)
- double complex clog (double complex z)

  Complex log function returns the natural logarithm of a complex number.

- float complex clogf (float complex z)
- __ldouble_t complex clogl (__ldouble_t complex z)
- double complex cpow (double complex x, double complex y)

  Complex power function returns x to power of y.

- float complex cpowf (float complex x, float complex y)
- __ldouble_t complex cpowl (__ldouble_t complex x, __ldouble_t complex y)
- double cabs (double complex z)

  Complex absolute value function.

- float cabsf (float complex z)
- __ldouble_t cabsl (__ldouble_t complex z)
- double complex csqrt (double complex z)

  Complex square root function.

- float complex csqrtf (float complex z)
- __ldouble_t complex csqrtl (__ldouble_t complex z)
- double carg (double complex z)

  Get the phase angle of the complex number z.

- float cargf (float complex z)
- __ldouble_t cargl (__ldouble_t complex z)
- double cimag (double complex z)

  Get the imaginary part of a complex number.

- float cimagf (float complex z)
- __ldouble_t cimagl (__ldouble_t complex z)
- double creal (double complex z)

  Get the real part of a complex number.

- float crealf (float complex z)
- __ldouble_t creall (__ldouble_t complex z)
- double complex conj (double complex z)

  Calculate the complex conjugate of a complex number.

- float complex conjf (float complex z)
- __ldouble_t complex conjl (__ldouble_t complex z)
- double complex cproj (double complex z)

  Project a point onto a a Riemann Sphere.

- float complex cprojf (float complex z)
- __ldouble_t complex cprojl (__ldouble_t complex z)

4.5.1 Detailed Description

Provides complex math API declarations. Complex numbers are numbers of the form z = a+b∗i, where a and b are real numbers and i = sqrt(-1), so that i∗i = -1.

There are other ways to represent that number. The pair (a,b) of real numbers may be viewed as a point in the plane, given by X- and Y-coordinates.

This same point may also be described by giving the pair of real numbers (r,phi), where r is the distance to the origin O, and phi the angle between the X-axis and the line Oz. That is: z = r∗exp(i∗phi) = r∗(cos(phi)+i∗sin(phi)).

The basic operations are defined on z = a+b∗i and w = c+d∗i as:

- addition: z+w = (a+c) + (b+d)∗i
- multiplication: z∗w = (a∗c - b∗d) + (a∗d + b∗c)∗i
- division: z/w = ((a∗c + b∗d)/(c∗c + d∗d)) + ((b∗c - a∗d)/(c∗c + d∗d))∗i

Nearly all math function have a complex counterpart but there are some complex-only functions.

Your C-compiler can work with complex numbers if it supports the C99 standard. Link with -lm. The imaginary unit is represented by I.

Most detailed descriptions of these functions come from BSD man pages. See file lib/LIB_LICENSE.txt for license info.

Definition in file complex.h.

4.5.2 Function Documentation

4.5.2.1 double cabs (double complex *z*)

Complex absolute value function.

The cabs() function returns the absolute value of the complex number z. The result is a real number.

Parameters

z Complex number to take the absolute value.

Returns

Real double absolute value of z. f(z) = |z|

4.5.2.2 float cabsf (float complex z)

See cabs()

4.5.2.3 __ldouble_t cabsl (__ldouble_t complex z)

See cabs()

4.5.2.4 double complex cacos (double complex z)

Complex inverse cosine function.

cacos(z) computes the inverse cosine of the complex floating-point number z, with branch cuts outside the interval [-1,1] along the real axis.

cacos() returns values in a strip of the complex plane with unbounded imaginary part, and real part in the interval [0, Pi].

For all complex floating point numbers z, cacos(conj(z)) = conj(cacos(z)).

The conjugate symmetry of cacos() is used to abbreviate the specification of special values.

- cacos(+-0 + 0i) returns Pi/2 - 0i.
- cacos(+-0 + NaN i) returns Pi/2 + NaN i.
- cacos(x + inf i) returns Pi/2 - inf i, for finite x.
- cacos(x + NaN i) returns NaN + NaN i, for finite nonzero x.
- cacos(-inf + yi) returns Pi - inf i, for finite positive-signed y.
- cacos(inf + yi) returns 0 - inf i, for finite positive-signed y.
- cacos(-inf + inf i) returns 3Pi/4 - inf i.
- cacos(inf + inf i) returns Pi/4 - inf i.
- cacos(+-inf + NaN i) returns NaN + inf i.
- cacos(NaN + yi) returns NaN + NaN i, for finite y.
- cacos(NaN + inf i) returns NaN - inf i.
- cacos(NaN + NaN i) returns NaN + NaN i.

Parameters

z Number to take inverse cosine from.

Returns

Inverse cosine of z.

4.5.2.5 float complex cacosf (float complex z)

See cacos()

4.5.2.6 double complex cacosh (double complex z)

complex inverse hyperbolic cosine function.

cacosh(z) computes the inverse hyperbolic cosine of the complex floating-point number z, with a branch cut on the interval [-inf, 1] along the real axis.

cacosh() returns values in a half-strip of the complex plane with positive real part and imaginary part in the interval [-Pi, Pi].

For all complex floating point numbers z, cacosh(conj(z)) = $*$ conj(cacosh(z)).

The conjugate symmetry of cacosh() is used to abbreviate the specification of special values.

- cacosh(+-0 + 0i) returns 0 + Pi/2 i.
- cacosh(x + inf i) returns inf + Pi/2 i, for finite x.
- cacosh(x + NaN i) returns NaN + NaN i, for finite nonzero x.
- cacosh(-inf + yi) returns inf + Pi i, for finite positive-signed y.
- cacosh(inf + yi) returns inf + 0i, for finite positive-signed y.
- cacosh(-inf + inf i) returns inf + 3Pi/4 i.
- cacosh(inf + inf i) returns inf + Pi/4 i.
- cacosh(+-inf + NaN i) returns inf + NaN i.
- cacosh(NaN + yi) returns NaN + NaN i, for finite y.
- cacosh(NaN + inf i) returns inf + NaN i.
- cacosh(NaN + NaN i) returns NaN + NaN i.

Parameters

z Complex number to calculate cacosh.

Returns

Complex inverse hyperbolic cosine of z.

4.5.2.7 float complex cacoshf (float complex z)

See cacosh()

4.5.2.8 __ldouble_t complex cacoshl (__ldouble_t complex z)

See cacosh()

4.5.2.9 __ldouble_t complex cacosl (__ldouble_t complex z)

See cacos()

4.5.2.10 double carg (double complex z)

Get the phase angle of the complex number z.

A complex number can be described by two real coordinates.

One may use rectangular coordinates and get f(x,y) = x + I ∗ y, such that x = creal(z) and y = cimag(z).

One may also use polar coordinates and get f(r,a) = r ∗ cexp(I ∗ a), such that r = cabs(z) or the "radius" of z, and a = carg(z) or the phase angle of z.

Parameters

z Number where angle is derived.

Returns

Real number representing the phase angle of the complex number z.

4.5.2.11 float cargf (float complex z)

See carg()

4.5.2.12 __ldouble_t cargl (__ldouble_t complex z)

See carg()

4.5.2.13 double complex casin (double complex z)

Complex inverse sine function.

casin(z) computes the inverse sine of the complex floating-point number z, with branch cuts outside the interval [-1, 1] on the real axis.

Function returns values in a strip of the complex plane with unbounded imaginary part, and real part in the interval [-Pi/2, Pi/2].

casin are defined in terms of the complex inverse hyperbolic functions as follows:

- casin(z) = -i ∗ casinh(i∗z),

Parameters

z Number for calculating the inverse sine.

Returns

Inverse sine of z.

4.5.2.14 float complex casinf (float complex z)

See casin()

4.5.2.15 double complex casinh (double complex z)

Complex inverse hyperbolic sine function.

casinh(z) computes the inverse hyperbolic sine of the complex floating-point number z, with branch cuts outside the interval [-i, i] along the imaginary axis.

A branch cut is a curve (with ends possibly open, closed, or half-open) in the complex plane across which an analytic multivalued function is discontinuous.

casinh() returns values in a strip of the complex plane with imaginary part in the interval [-Pi/2, Pi/2].

For all complex floating point numbers z,

- casinh(conj(z)) = conj(casinh(z)).
- casinh(-z) = -casinh(z)

Parameters

z Complex number to calculate casinh.

Returns

Complex inverse hyperbolic sine of z.

4.5.2.16 float complex casinhf (float complex z)

See casinh()

4.5.2.17 __ldouble_t complex casinhl (__ldouble_t complex z)

See casinh()

4.5.2.18 __ldouble_t complex casinl (__ldouble_t complex z)

See casin()

4.5.2.19 double complex catan (double complex z)

Complex inverse tangent function.

ctan(z) computes the inverse tangent of the complex floating-point number z, with branch cuts outside the interval [-i, i] on the imaginary axis.

Function returns values in a strip of the complex plane with unbounded imaginary part, and real part in the interval [-Pi/2, Pi/2].

catan is defined in terms of the complex inverse hyperbolic function as follows:

- catan(z) = -i $*$ catanh(i$*$z).

Parameters

z Number for calculating the inverse tangent.

Returns

Inverse tan of z.

4.5.2.20 float complex catanf (float complex *z*)

See catan()

4.5.2.21 double complex catanh (double complex *z*)

Complex inverse hyperbolic tangent function.

catanh(z) computes the inverse hyperbolic tangent of the complex floating-point number z, with branch cuts outside the interval [-1, 1] along the real axis.

A branch cut is a curve (with ends possibly open, closed, or half-open) in the complex plane across which an analytic multivalued function is discontinuous.

catanh() returns values in a strip of the complex plane with imaginary part in the interval [-Pi/2, Pi/2].

For all complex floating point numbers z,

- catanh(conj(z)) = conj(catanh(z))
- catanh(-z) = -catanh(z)

Parameters

z Complex number to calculate catanh.

Returns

Complex inverse hyperbolic tangent of z.

4.5.2.22 float complex catanhf (float complex *z*)

See catanh()

4.5.2.23 __ldouble_t complex catanhl (__ldouble_t complex *z*)

See catanh()

4.5.2.24 __ldouble_t complex catanl (__ldouble_t complex *z*)

See catan()

4.5.2.25 double complex ccos (double complex *z*)

Complex cosine function.

Computes the cosine of the complex floating-point number z.

- ccos(z) = ccosh(i∗z)

Parameters

z Number for calculating cosine.

Returns

complex cosine of z.

4.5.2.26 float complex ccosf (float complex *z*)

See ccos()

4.5.2.27 double complex ccosh (double complex *z*)

Complex hyperbolic cosine function.

ccos(z) computes the hyperbolic cosine of the complex floating-point number z.

For all complex floating point numbers z,

- ccosh(conj(z)) = conj(ccosh(z))
- ccosh(-z) = ccosh(z)

The symmetries of ccosh() are used to abbreviate the specification of special values.

- ccosh(0 + 0i) returns 1 + 0i.
- ccosh(0 + inf i) returns NaN + 0i, and raises the invalid flag.
- ccosh(0 + NaN i) returns NaN + 0i.
- ccosh(x + inf i) returns NaN + NaN i, and raises the invalid flag, for finite nonzero x.
- ccosh(x + NaN i) returns NaN + NaN i, for finite nonzero x.
- ccosh(inf + 0i) returns inf + 0i.
- ccosh(inf + yi) returns inf ∗ cis(y), for finite positive y, where cis(y) = cos(y) + i∗sin(y).
- ccosh(inf + inf i) returns inf + NaN i, and raises the invalid flag.
- ccosh(inf + NaN i) returns inf + NaN i.
- ccosh(NaN + 0i) returns NaN + 0i.
- ccosh(NaN + yi) returns NaN + NaN i, for nonzero numbers y.
- ccosh(NaN + NaN i) returns NaN + NaN i.

Parameters

z Complex number to calculate ccosh.

Returns

Complex hyperbolic cosine of z.

4.5.2.28 float complex ccoshf (float complex z)

See ccosh()

4.5.2.29 __ldouble_t complex ccoshl (__ldouble_t complex z)

See ccosh()

4.5.2.30 __ldouble_t complex ccosl (__ldouble_t complex z)

See ccos()

4.5.2.31 double complex cexp (double complex z)

Complex expotential function returns e raised to z power.

This function calculates e (2.71828..., the base of natural logarithms) raised to the power of z.

In other words: e ** z such that ** means raise to power of.

Parameters

z The power to raise e.

Returns

A double complex e to the power of z. f(z) = e ** z

4.5.2.32 float complex cexpf (float complex z)

See cexp()

4.5.2.33 __ldouble_t complex cexpl (__ldouble_t complex z)

See cexp()

4.5.2.34 double cimag (double complex z)

Get the imaginary part of a complex number.

This function returns the imaginary part of the complex number z. Such that f(z) = creal(c) + I * cimag(z)

Parameters

z An complex number.

Returns

A double representing the imaginary part of z.

4.5.2.35 float cimagf (float complex z)

See cimag()

4.5.2.36 __ldouble_t cimagl (__ldouble_t complex z)

See cimag()

4.5.2.37 double complex clog (double complex z)

Complex log function returns the natural logarithm of a complex number.

The logarithm clog() is the inverse function of the exponential cexp(). Thus, if y = clog(z), then z = cexp(y). The imaginary part of y is chosen in the interval [-pi,pi].

Parameters

z Number to calculate the natuarl log.

Returns

A double complex natural log of z. f(z) = ln(z)

4.5.2.38 float complex clogf (float complex z)

See clog()

4.5.2.39 __ldouble_t complex clogl (__ldouble_t complex z)

See clog()

4.5.2.40 double complex conj (double complex z)

Calculate the complex conjugate of a complex number.

The conj() function returns the complex conjugate value of z. That is the value obtained by changing the sign of the imaginary part.

Parameters

z A complex number

Returns

The complex conjugate of the number.

4.5.2.41 float complex conjf (float complex z)

See conj()

4.5.2.42 __ldouble_t complex conjl (__ldouble_t complex z)

See conj()

4.5.2.43 double complex cpow (double complex x, double complex y)

Complex power function returns x to power of y.

The function calculates x raised to the power y with a branch cut for x along the negative real axis.

A branch cut is a curve (with ends possibly open, closed, or half-open) in the complex plane across which an analytic multivalued function is discontinuous.

Parameters

x Number to raise.

y Power to raise x.

Returns

A double complex x to the power of y. f(x,y) = x ** y

4.5.2.44 float complex cpowf (float complex x, float complex y)

See cpow()

4.5.2.45 __ldouble_t complex cpowl (__ldouble_t complex x, __ldouble_t complex y)

See cpow()

4.5.2.46 double complex cproj (double complex z)

Project a point onto a a Riemann Sphere.

This function projects a point in the plane onto the surface of a Riemann Sphere, the one-point compactification of the complex plane. Each finite point z projects to z itself.

Every complex infinite value is projected to a single infinite value, namely to positive infinity on the real axis.

Parameters

z A complex number.

Returns

A single point on the Riemann sphere.

4.5.2.47 float complex cprojf (float complex z)

See cproj()

4.5.2.48 __ldouble_t complex cprojl (__ldouble_t complex z)

See cproj()

4.5.2.49 double creal (double complex z)

Get the real part of a complex number.

This function returns the real part of the complex number z. Such that f(z) = creal(c) + I ∗ cimag(z)

Parameters

z A complex number.

Returns

A double representing the real part of z.

4.5.2.50 float crealf (float complex z)

See creal()

4.5.2.51 __ldouble_t creall (__ldouble_t complex z)

See creal()

4.5.2.52 double complex csin (double complex z)

Complex sine function.

Computes the sine of the complex floating-point number z.

- csin(z) = -i ∗ csinh(i∗z)

Parameters

z Number for calculating sine.

Returns

complex sine of z.

4.5.2.53 float complex csinf (float complex z)

See csin()

4.5.2.54 double complex csinh (double complex z)

Complex hyperbolic sine function.

csin(z) computes the hyperbolic sine of the complex floating-point number z.

For all complex floating point numbers z,

- csinh(conj(z)) = conj(csinh(z))
- csinh(-z) = -csinh(z)

The symmetries of csinh() are used to abbreviate the specification of special values.

- csinh(0 + 0i) returns 0 + 0i.
- csinh(0 + inf i) returns 0 + NaN i, and raises the invalid flag.
- csinh(0 + NaN i) returns 0 + NaN i.
- csinh(x + inf i) returns NaN + NaN i, and raises the invalid flag, for finite nonzero x.
- csinh(x + NaN i) returns NaN + NaN i, for finite nonzero x.
- csinh(inf + 0i) returns inf + 0i.
- csinh(inf + yi) returns inf ∗ cis(y), for finite positive y, where cis(y) = cos(y) + i∗sin(y).
- csinh(inf + inf i) returns inf + NaN i, and raises the invalid flag.
- csinh(inf + NaN i) returns inf + NaN i.
- csinh(NaN + 0i) returns NaN + 0i.
- csinh(NaN + yi) returns NaN + NaN i, for nonzero numbers y.
- csinh(NaN + NaN i) returns NaN + NaN i.

Parameters

z Complex number to calculate csinh.

Returns

Complex hyperbolic sine of z.

4.5.2.55 float complex csinhf (float complex *z*)

See csinh()

4.5.2.56 __ldouble_t complex csinhl (__ldouble_t complex *z*)

See csinh()

4.5.2.57 __ldouble_t complex csinl (__ldouble_t complex *z*)

See csin()

4.5.2.58 double complex csqrt (double complex z)

Complex square root function.

Calculate the square root of a given complex number, with nonnegative real part, and with a branch cut along the negative real axis. That means that csqrt(-1+eps∗I) will be close to I while csqrt(-1-eps∗I) will be close to -I, if eps is a small positive real number.

Parameters

z Complex number to take square root.

Returns

Complex square root of z.

4.5.2.59 float complex csqrtf (float complex z)

See csqrt()

4.5.2.60 __ldouble_t complex csqrtl (__ldouble_t complex z)

See csqrt()

4.5.2.61 double complex ctan (double complex z)

Complex tangent function.

Computes the tangent of the complex floating-point number z.

- ctan(z) = -i ∗ ctanh(i∗z)

Parameters

z Number for calculating tangent.

Returns

complex tangent of z.

4.5.2.62 float complex ctanf (float complex z)

See ctan()

4.5.2.63 double complex ctanh (double complex z)

Complex hyperbolic tangent function.

ctanh(z) computes the hyperbolic tangent of the complex floating-point number z.

For all complex floating point numbers z,

- ctanh(conj(z)) = conj(ctanh(z)),
- ctanh(-z) = -ctanh(z).

The symmetries of ctanh() are used to abbreviate the specification of special values.

- ctanh(0 + 0i) returns 0 + 0i.
- ctanh(0 + inf i) returns NaN + NaN i, and raises the invalid flag.
- ctanh(0 + NaN i) returns NaN + NaN i.
- ctanh(x + inf i) returns NaN + NaN i, and raises the invalid flag, for finite nonzero x.
- ctanh(x + NaN i) returns NaN + NaN i, for finite nonzero x.
- ctanh(inf + 0i) returns 1 + 0i.
- ctanh(inf + yi) returns 1 +- 0i, for finite positive y, with the sign chosen to match the sign of sin(2y).
- ctanh(inf + inf i) returns 1 + 0i.
- ctanh(inf + NaN i) returns 1 + 0i.
- ctanh(NaN + 0i) returns NaN + 0i.
- ctanh(NaN + yi) returns NaN + NaN i, for nonzero numbers y.
- ctanh(NaN + NaN i) returns NaN + NaN i.

Parameters

z Complex number to calculate ctanh.

Returns

Complex hyperbolic tangent of z.

4.5.2.64 float complex ctanhf (float complex *z*)

See ctanh()

4.5.2.65 __ldouble_t complex ctanhl (__ldouble_t complex *z*)

See ctanh()

4.5.2.66 __ldouble_t complex ctanl (__ldouble_t complex *z*)

See ctan()

4.6 include/ctype.h File Reference

Provides character class check API macros.

```
#include <sys/ctype.h>
```

Defines

- #define isalnum(c) __isctype(c, _CTalnum)
- #define isalpha(c) __isctype(c, _CTalpha)
- #define isblank(c) __isctype(c, _CTb)
- #define iscntrl(c) __isctype(c, _CTc)
- #define isdigit(c) __isctype(c, _CTd)
- #define isgraph(c) (!__isctype(c, (_CTc|_CTs)) && (__ctype_get(c) != 0))
- #define islower(c) __isctype(c, _CTl)
- #define isprint(c) (!__isctype(c, (_CTc)) && (__ctype_get(c) != 0))
- #define ispunct(c) __isctype(c, _CTp)
- #define isspace(c) __isctype(c, _CTs)
- #define isupper(c) __isctype(c, _CTu)
- #define isxdigit(c) __isctype(c, _CTx)

Functions

- int tolower (int c)
- int toupper (int c)

4.6.1 Detailed Description

Provides character class check API macros. These functions check whether c, which must have the value of an unsigned char or EOF, falls into a certain character class according to the current locale.

Conforms to C99, 4.3BSD. C89 specifies all of these functions except isascii() and isblank(). isascii() is a BSD extension and is also an SVr4 extension. isblank() conforms to POSIX.1-2001 and C99 7.4.1.3. POSIX.1-2008 marks isascii() as obsolete, noting that it cannot be used portably in a localized application.

Definition in file ctype.h.

4.6.2 Define Documentation

4.6.2.1 #define isalnum(*c*) __isctype(c, _CTalnum)

Checks for an alphanumeric class character.

Definition at line 57 of file ctype.h.

4.6.2.2 #define isalpha(*c*) __isctype(c, _CTalpha)

Checks for an alphabetic class character.

The isalpha function tests for any character for which isupper or islower is true, or any character that is one of a locale-specific set of alphabetic characters for which none of iscntrl, isdigit, ispunct, or isspace is true.156) In the "C" locale, isalpha returns true only for the characters for which isupper or islower is true.

Definition at line 67 of file ctype.h.

4.6.2.3 #define isblank(c) __isctype(c, _CTb)

Checks for a blank (space or tab) character.

Definition at line 71 of file ctype.h.

4.6.2.4 #define iscntrl(c) __isctype(c, _CTc)

Checks for a control character.

Definition at line 75 of file ctype.h.

4.6.2.5 #define isdigit(c) __isctype(c, _CTd)

Checks for a digit (0 through 9) character.

Definition at line 79 of file ctype.h.

4.6.2.6 #define isgraph(c) (!__isctype(c, (_CTc|_CTs)) && (__ctype_get(c) != 0))

Checks for a any printable character except for space.

Definition at line 83 of file ctype.h.

4.6.2.7 #define islower(c) __isctype(c, _CTl)

Checks for a lower-case character.

The islower function tests for any character that is a lowercase letter or is one of a locale-specific set of characters for which none of iscntrl, isdigit, ispunct, or isspace is true. In the ∗ "C" locale, islower returns true only for the characters defined as lowercase letters.

Definition at line 93 of file ctype.h.

4.6.2.8 #define isprint(c) (!__isctype(c, (_CTc)) && (__ctype_get(c) != 0))

Checks for any printable character including space.

Definition at line 97 of file ctype.h.

4.6.2.9 #define ispunct(c) __isctype(c, _CTp)

Checks for any printable character that is not a space or alphanumeric character.

The ispunct function tests for any printing character that is one of a locale-specific set of punctuation characters for which neither isspace nor isalnum is true.

Definition at line 105 of file ctype.h.

4.6.2.10 #define isspace(c) __isctype(c, _CTs)

Checks for a white space character: space, \f, \n, \r, \t, \v in "C" locales.

Definition at line 109 of file ctype.h.

4.6.2.11 #define isupper(c) __isctype(c, _CTu)

Checks for an upper-case character.

The isupper function tests for any character that is an uppercase letter or is one of a locale-specific set of characters for which none of iscntrl, isdigit, ispunct, or isspace is true. In the "C" locale, isupper returns true only for the characters defined as uppercase letters.

Definition at line 119 of file ctype.h.

4.6.2.12 #define isxdigit(c) __isctype(c, _CTx)

Checks for a hexadecimal digit: 0 through 9, a through f, or A through F.

Definition at line 123 of file ctype.h.

4.6.3 Function Documentation

4.6.3.1 int tolower (int c)

Converts an uppercase letter to a corresponding lowercase letter.

If the argument is a character for which isupper is true and there are one or more corresponding characters, as specified by the current locale, for which islower is true, the tolower function returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

Parameters

c Letter to convert.

Returns

Uppercase version of c according to locale.

4.6.3.2 int toupper (int c)

Converts a lowercase letter to a corresponding uppercase letter.

If the argument is a character for which islower is true and there are one or more corresponding characters, as specified by the current locale, for which isupper is true, the toupper function returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

Parameters

c Letter to convert.

Returns

Lowercase version of c according to locale.

4.7 include/sys/ctype.h File Reference

Defines macros and types used by include/ctype.h.

4.7.1 Detailed Description

Defines macros and types used by include/ctype.h.

Definition in file ctype.h.

4.8 include/dirent.h File Reference

Provides directory operations API declarations.

Classes

- struct DIRstruct
  Defines the dirent.h DIR struct.
- struct dirent
  Defines the dirent.h dirent struct.

Typedefs

- typedef struct DIRstruct DIR
  Defines the dirent.h DIR struct.

Functions

- DIR * opendir (const char *path)
- struct dirent * readdir (DIR *dirp)
- int closedir (DIR *dirp)

4.8.1 Detailed Description

Provides directory operations API declarations.

Definition in file dirent.h.

4.8.2 Typedef Documentation

4.8.2.1 typedef struct DIRstruct DIR

Defines the dirent.h DIR struct.

DIR fields:

- uint32_t currentcluster; // current cluster in dir
- uint8_t currentsector; // current sector in cluster
- uint8_t currententry; // current dir entry in sector

- uint8_t *scratch; // ptr to user-supplied sector buffer
- uint8_t flags; // internal DOSFS flags

4.8.3 Function Documentation

4.8.3.1 int closedir (DIR * *dirp*)

Closes a directory.

The closedir() function closes the named directory stream and frees the structure associated with the dirp pointer, returning 0 on success. On failure, -1 is returned and the global variable errno is set to indicate the error.

Parameters

dirp Pointer to directory to close.

Returns

Zero on success or -1 on failure and sets errno.

4.8.3.2 DIR* opendir (const char * *path*)

Opens a directory.

The opendir() function opens the directory named by dirname, associates a directory stream with it, and returns a pointer to be used to identify the directory stream in subsequent operations. The pointer NULL is returned if dirname cannot be accessed or if it cannot malloc(3) enough memory to hold the whole thing.

Parameters

path The directory to be opened.

Returns

DIR struct pointer to be used in subsequent operations. Returns NULL on error.

4.8.3.3 struct dirent* readdir (DIR * *dirp*) [read]

Returns pointer to the next directory entry.

The readdir() function returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid seekdir() operation.

Sample code searches a directory for entry "name":

```
len = strlen(name);
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL) {
  if (!strcmp(dp->d_name, name)) {
    closedir(dirp);
    return FOUND;
  }
}
(void)closedir(dirp);
return NOT_FOUND;
```

Precondition

Parameter dirp must be successfully opened by opendir.

Parameters

`[in,out]` ***dirp*** Pointer to the DIR struct.

Returns

Pointer to the next directory entry, or NULL if end of directory.

4.9 include/driver.h File Reference

Provides driver API functions for initializing devices.

```
#include <stdio.h>
#include <sys/driver.h>
```

Functions

- void _InitIO (void)

4.9.1 Detailed Description

Provides driver API functions for initializing devices. Copyright (c) 2011-2012 by Parallax, Inc. All rights MIT Licensed.

The _InitIO() function is called at program startup before main().

It initializes stdio devices as defined in include/sys/driver.h

New devices having stdio interfaces can be added to the driver table.

Definition in file driver.h.

4.9.2 Function Documentation

4.9.2.1 void _InitIO (void)

See also

Details above

4.10 include/sys/driver.h File Reference

Contains driver API for stdio devices.

Classes

- struct __driver

  Generic and customizable driver struct for stdio devices.

Typedefs

- typedef struct __driver _Driver

Functions

- int _null_read (FILE *fp, unsigned char *buf, int size)
- int _null_write (FILE *fp, unsigned char *buf, int size)
- int _term_read (FILE *fp, unsigned char *buf, int size)
- int _term_write (FILE *fp, unsigned char *buf, int size)

Variables

- _Driver * _driverlist [ ]

  Device driver array of __driver structs.

4.10.1 Detailed Description

Contains driver API for stdio devices.

Definition in file driver.h.

4.10.2 Typedef Documentation

4.10.2.1 typedef struct __driver _Driver

Typedef for the _Driver struct

This struct is used by stdio. It lets a program define custom devices.

See also

__driver for more information on custom devices.

Note

This structure is included by <stdio.h>, so we cannot pollute the namespace with definitions; use underscores in front of names

Definition at line 24 of file driver.h.

4.10.3 Function Documentation

4.10.3.1 int _null_read (FILE * *fp*, unsigned char * *buf*, int *size*)

Use this when no other read function is applicable

4.10.3.2 int _null_write (FILE * *fp*, unsigned char * *buf*, int *size*)

Use this when no other write function is applicable

4.10.3.3 int _term_read (FILE * *fp*, unsigned char * *buf*, int *size*)

Use as getbyte function to do cooked I/O input

4.10.3.4 int _term_write (FILE * *fp*, unsigned char * *buf*, int *size*)

Use as putbyte function to do cooked I/O output

4.10.4 Variable Documentation

4.10.4.1 _Driver* _driverlist[ ]

Device driver array of __driver structs.

See also

Detailed Description in include/sys/driver.h

4.11 include/errno.h File Reference

Defines error condition reporting macros.

```
#include <sys/thread.h>
```

Defines

- #define EOK 0
- #define EDOM 1
- #define ERANGE 2
- #define EILSEQ 3
- #define ENOENT 4
- #define EBADF 5
- #define EACCES 6
- #define ENOMEM 7
- #define EAGAIN 8
- #define EEXIST 9
- #define EINVAL 10
- #define EMFILE 11
- #define EIO 12
- #define ENOTDIR 13
- #define EISDIR 14
- #define EROFS 15
- #define ENOSYS 16
- #define ENOTEMPTY 17
- #define ENAMETOOLONG 18
- #define ENOSEEK 19
- #define EFAULT 20
- #define EPIPE 21
- #define EBUSY 22

- #define EXDEV 23
- #define ENOSPC 24
- #define EINTR 25
- #define EWOULDBLOCK EAGAIN
- #define ESPIPE EPIPE
- #define ENOEXEC ENOSYS
- #define EFBIG ERANGE
- #define EOPNOTSUPP ENOSYS
- #define ENOTTY ENOSEEK
- #define EPERM EACCES

4.11.1 Detailed Description

Defines error condition reporting macros. These simple macros support error conditions for the libraries. The only macros required by ANSI-C are EDOM, EILSEQ, and ERANGE.

The undocumented "Unlikely" and "Network" macros are included for compatibility with GNU libstdc++.

Definition in file errno.h.

4.11.2 Define Documentation

4.11.2.1 #define EACCES 6

Permission denied

Definition at line 42 of file errno.h.

4.11.2.2 #define EAGAIN 8

Temporary failure

Definition at line 48 of file errno.h.

4.11.2.3 #define EBADF 5

Bad file number

Definition at line 39 of file errno.h.

4.11.2.4 #define EBUSY 22

Device or resource busy

Definition at line 90 of file errno.h.

4.11.2.5 #define EDOM 1

Math arg out of domain of func

Definition at line 27 of file errno.h.

4.11.2.6 #define EEXIST 9

File exists

Definition at line 51 of file errno.h.

4.11.2.7 #define EFAULT 20

Bad address

Definition at line 84 of file errno.h.

4.11.2.8 #define EFBIG ERANGE

Math result not representable

Definition at line 155 of file errno.h.

4.11.2.9 #define EILSEQ 3

Illegal sequence

Definition at line 33 of file errno.h.

4.11.2.10 #define EINTR 25

System call interrupted

Definition at line 99 of file errno.h.

4.11.2.11 #define EINVAL 10

Invalid argument

Definition at line 54 of file errno.h.

4.11.2.12 #define EIO 12

I/O error

Definition at line 60 of file errno.h.

4.11.2.13 #define EISDIR 14

Is a directory

Definition at line 66 of file errno.h.

4.11.2.14 #define EMFILE 11

Too many open files

Definition at line 57 of file errno.h.

4.11.2.15 #define ENAMETOOLONG 18

File or path name too long

Definition at line 78 of file errno.h.

4.11.2.16 #define ENOENT 4

No such file or directory

Definition at line 36 of file errno.h.

4.11.2.17 #define ENOEXEC ENOSYS

Function not implemented

Definition at line 152 of file errno.h.

4.11.2.18 #define ENOMEM 7

Not enough core

Definition at line 45 of file errno.h.

4.11.2.19 #define ENOSEEK 19

Device not seekable

Definition at line 81 of file errno.h.

4.11.2.20 #define ENOSPC 24

No space on device

Definition at line 96 of file errno.h.

4.11.2.21 #define ENOSYS 16

Function not implemented

Definition at line 72 of file errno.h.

4.11.2.22 #define ENOTDIR 13

Not a directory

Definition at line 63 of file errno.h.

4.11.2.23 #define ENOTEMPTY 17

Directory not empty

Definition at line 75 of file errno.h.

4.11.2.24 #define ENOTTY ENOSEEK

Device not seekable

Definition at line 161 of file errno.h.

4.11.2.25 #define EOK 0

Undefined error

Definition at line 24 of file errno.h.

4.11.2.26 #define EOPNOTSUPP ENOSYS

Function not implemented

Definition at line 158 of file errno.h.

4.11.2.27 #define EPERM EACCES

Permission denied

Definition at line 164 of file errno.h.

4.11.2.28 #define EPIPE 21

Broken pipe

Definition at line 87 of file errno.h.

4.11.2.29 #define ERANGE 2

Math result not representable

Definition at line 30 of file errno.h.

4.11.2.30 #define EROFS 15

Read only file system

Definition at line 69 of file errno.h.

4.11.2.31 #define ESPIPE EPIPE

Broken pipe

Definition at line 149 of file errno.h.

4.11.2.32 #define EWOULDBLOCK EAGAIN

Temporary failure

Definition at line 146 of file errno.h.

4.11.2.33 #define EXDEV 23

Cross device link

Definition at line 93 of file errno.h.

4.12 include/fcntl.h File Reference

Provides fcntl() interface API macros.

Defines

- #define O_RDONLY 0
- #define O_WRONLY 1
- #define O_RDWR 2
- #define O_CREAT 4
- #define O_TRUNC 8
- #define O_EXCL 16
- #define FD_CLOEXEC 0x100
- #define F_SETFD 0x200

4.12.1 Detailed Description

Provides fcntl() interface API macros. The fcntl() function is only provided in this library to support libstdc++ std namespace.

The fcntl() function is not really useful as Propeller-GCC does not implement fcntl().

Definition in file fcntl.h.

4.12.2 Define Documentation

4.12.2.1 #define F_SETFD 0x200

Definition provided for convenience and libstdc++ build only. Propeller-GCC does not implement fcntl().

Set the file descriptor flags to the value specified by arg.

Definition at line 49 of file fcntl.h.

4.12.2.2 #define FD_CLOEXEC 0x100

Definition provided for convenience and libstdc++ build only. Propeller-GCC does not allow forking applications.

If bit is 0, the file descriptor will remain open across execve(2), otherwise it will be closed.

Definition at line 41 of file fcntl.h.

4.12.2.3 #define O_CREAT 4

Set open file mode to create

Definition at line 26 of file fcntl.h.

4.12.2.4 #define O_EXCL 16

Set open file mode to exclusive

Definition at line 32 of file fcntl.h.

4.12.2.5 #define O_RDONLY 0

Set open file status to read only

Definition at line 17 of file fcntl.h.

4.12.2.6 #define O_RDWR 2

Set open file status to read-write

Definition at line 23 of file fcntl.h.

4.12.2.7 #define O_TRUNC 8

Set open file mode to truncate

Definition at line 29 of file fcntl.h.

4.12.2.8 #define O_WRONLY 1

Set open file status to write only

Definition at line 20 of file fcntl.h.

4.13 include/fenv.h File Reference

Provides floating point exception declarations.

`#include <sys/thread.h>`

Defines

- #define FE_DFL_ENV (&__dflt_fenv)
- #define FE_ALL_EXCEPT 0
- #define FE_TONEAREST 0

Typedefs

- typedef unsigned char fexcept_t
- typedef _fenv_t fenv_t

Functions

- int feclearexcept (int excepts)
- int fegetexceptflag (fexcept_t *flagp, int excepts)
- int feraiseexcept (int excepts)
- int fesetexceptflag (const fexcept_t *flagp, int excepts)
- int fetestexcept (int excepts)
- int fegetround (void)
- int fesetround (int round)
- int fegetenv (fenv_t *envp)
- int feholdexcept (fenv_t *envp)
- int fesetenv (const fenv_t *envp)
- int feupdateenv (const fenv_t *envp)

4.13.1 Detailed Description

Provides floating point exception declarations. The descriptions here are only provided for completeness as the functions are not intended to be called by normal user programs.

The header <fenv.h> declares two types and several macros and functions to provide access to the floating-point environment. The floating-point environment refers collectively to any floating-point status flags and control modes supported by the implementation.160) A floating-point status flag is a system variable whose value is set as a side effect of floating-point arithmetic to provide auxiliary information. A floatingpoint control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.

Definition in file fenv.h.

4.13.2 Define Documentation

4.13.2.1 #define FE_ALL_EXCEPT 0

no floating point exceptions are defined

Definition at line 50 of file fenv.h.

4.13.2.2 #define FE_DFL_ENV (&__dflt_fenv)

Represents the default floating-point environment — the one installed at program startup — and has type pointer to const-qualified fenv_t. It can be used as an argument to <fenv.h> functions that manage the floating-point environment.

Definition at line 47 of file fenv.h.

4.13.2.3 #define FE_TONEAREST 0

only round-to-nearest mode supported

Definition at line 53 of file fenv.h.

4.13.3 Typedef Documentation

4.13.3.1 typedef _fenv_t fenv_t

Represents the entire floating-point environment.

Definition at line 37 of file fenv.h.

4.13.3.2 typedef unsigned char fexcept_t

Represents the floating-point exception flags collectively, including any status the implementation associates with the flags.

Definition at line 32 of file fenv.h.

4.13.4 Function Documentation

4.13.4.1 int feclearexcept (int *excepts*)

Clears the supported exceptions requested by its argument.

4.13.4.2 int fegetenv (fenv_t * *envp*)

Stores the current floating-point environment in the object pointed to by envp.

4.13.4.3 int fegetexceptflag (fexcept_t * *flagp,* int *excepts*)

The fegetexceptflag function stores an implementation-defined representation of the exception flags indicated by the argument excepts in the object pointed to by the argument flagp.

4.13.4.4 int fegetround (void)

The fegetround function gets the current rounding direction.

4.13.4.5 int feholdexcept (fenv_t * *envp*)

The feholdexcept function saves the current floating-point environment in the object pointed to by envp, clears the exception flags, and then installs a non-stop (continue on exceptions) mode, if available, for all exceptions.

4.13.4.6 int feraiseexcept (int *excepts*)

The feraiseexcept function raises the supported exceptions represented by its argument. The order in which these exceptions are raised is unspecified, except as stated in F.7.6. Whether the feraiseexcept function additionally raises the inexact exception whenever it raises the overflow or underflow exception is implementation defined.

4.13.4.7 int fesetenv (const fenv_t * *envp*)

The fesetenv function establishes the floating-point environment represented by the object pointed to by envp. The argument envp shall point to an object set by a call to fegetenv or feholdexcept, or equal the macro FE_DFL_ENV or an implementation-defined environment macro. Note that fesetenv merely installs the state of the exception flags represented through its argument, and does not raise these exceptions.

4.13.4.8 int fesetexceptflag (const fexcept_t * *flagp,* int *excepts*)

The fesetexceptflag function sets the complete status for those exception flags indicated by the argument excepts, according to the representation in the object pointed to by flagp. The value of *flagp shall have been set by a previous call to fegetexceptflag whose second argument represented at least those exceptions represented by the argument excepts. This function does not raise exceptions, but only sets the state of the flags.

4.13.4.9 int fesetround (int *round*)

The fesetround function establishes the rounding direction represented by its argument round. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

4.13.4.10 int fetestexcept (int *excepts*)

The fetestexcept function determines which of a specified subset of the exception flags are currently set. The excepts argument specifies the exception flags to be queried.

4.13.4.11 int feupdateenv (const fenv_t * *envp*)

The feupdateenv function saves the currently raised exceptions in its automatic storage, installs the floating-point environment represented by the object pointed to by envp, and then raises the saved exceptions. The argument envp shall point to an object set by a call to feholdexcept or fegetenv, or equal the macro FE_DFL_ENV or an implementation-defined environment macro.

4.14 include/float.h File Reference

Provides characteristics of floating types.

4.14.1 Detailed Description

Provides characteristics of floating types. Please see source for comments.

Definition in file float.h.

4.15 include/inttypes.h File Reference

Provides API for format conversion of integer types.

```
#include <stdint.h>
```

Functions

- intmax_t strtoimax (const char *__restrict, char **__restrict, int)
- uintmax_t strtouimax (const char *__restrict, char **__restrict, int)
- intmax_t wcstoimax (const _WCHAR_T_TYPE *__restrict, _WCHAR_T_TYPE **__restrict, int)
- uintmax_t wcstouimax (const _WCHAR_T_TYPE *__restrict, _WCHAR_T_TYPE **__restrict, int)

4.15.1 Detailed Description

Provides API for format conversion of integer types. It declares four functions for converting numeric character strings to greatest-width integers and, for each type declared in <stdint.h>, it defines corresponding macros for conversion specifiers for use with the formatted input/output functions.

These macros are provided mainly in support of fprintf and fscanf.

Definition in file inttypes.h.

4.15.2 Function Documentation

4.15.2.1 intmax_t strtoimax (const char * *__restrict*, char ** *__restrict*, int)

The strtoimax and strtoumax functions are equivalent to the strtol, strtoll, strtoul, and strtoull functions, except that the initial portion of the string is converted to intmax_t and uintmax_t representation, respectively.

Returns

The strtoimax and strtoumax functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, INTMAX_MAX, INTMAX_MIN, or UINTMAX_MAX is returned (according to the return type and sign of the value, if any), and the value of the macro ERANGE is stored in errno.

4.15.2.2 uintmax_t strtouimax (const char * *__restrict*, char ** *__restrict*, int)

see strtoimax

4.15.2.3 intmax_t wcstoimax (const _WCHAR_T_TYPE * *__restrict*, _WCHAR_T_TYPE ** *__restrict*, int)

The wcstoimax and wcstoumax functions are equivalent to the wcstol, wcstoll, wcstoul, and wcstoull functions except that the initial portion of the wide string is converted to intmax_t and uintmax_t representation,

respectively.

Returns

The wcstoimax function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, INTMAX_MAX, INTMAX_MIN, or UINTMAX_MAX is returned (according to the return type and sign of the value, if any), and the value of the macro ERANGE is stored in errno.

4.15.2.4 uintmax_t wcstouimax (const _WCHAR_T_TYPE * *__restrict,* _WCHAR_T_TYPE ** *__restrict,* int)

see wcstrtoimax

4.16 include/iso646.h File Reference

Provides macros for keyboards not having equivalent symbols.

4.16.1 Detailed Description

Provides macros for keyboards not having equivalent symbols. The iso646.h header file is part of the C standard library. It was added to this library in a 1995 amendment to the C90 standard. It defines a number of macros which allow programmers to use C language bitwise and logical operators, which, without the header file, cannot be quickly or easily typed on some international and non-QWERTY keyboards.

Definition in file iso646.h.

4.17 include/limits.h File Reference

Defines macros for limits and parameters of the standard integer types.

```
#include <compiler.h>
#include <sys/syslimits.h>
```

4.17.1 Detailed Description

Defines macros for limits and parameters of the standard integer types. The compiler supports data with the following characteristics:

- 8 bit char
- 16 bit short
- 32 bit int
- 32 bit long
- 64 bit long long

The compiler supports UTF-8, but does not support multi-byte characters.

Definition in file limits.h.

4.18 include/locale.h File Reference

Provides multi-lingual localization API functions.

```
#include <compiler.h>
#include <sys/null.h>
```

Classes

- struct lconv

  lconv contains members related to the formatting of numeric values.

Defines

- #define LC_ALL 0x001F

  Selects category names for the entire locale.

- #define LC_COLLATE 0x0001

  Affects the behavior of strcoll and strxfrm functions.

- #define LC_CTYPE 0x0002

  Affects the behavior of the character handling and multibyte functions.

- #define LC_MONETARY 0x0004

  Affects the monetary format information returned by localeconv.

- #define LC_NUMERIC 0x0008

  Affects the decimal point character and other information for formatted I/O.

- #define LC_TIME 0x0010

  Affects the behavior of the strftime and strfxtime functions.

Functions

- char * setlocale (int category, const char *locale)
- struct lconv * localeconv (void)

4.18.1 Detailed Description

Provides multi-lingual localization API functions. C localization functions are used in multilingual programs to adapt to the specific locale for displaying numbers and currency. Functions affect the C Standard Library I/O function behavior.

The only locales supported is "C" and a C.UTF-8 locale which is the same as C but supports UTF-8 multibyte character encoding. Things like currency and so forth are hard coded to the C locale.

Definition in file locale.h.

4.18.2 Function Documentation

4.18.2.1 struct lconv* localeconv (void) [read]

Returns

An object of type struct lconv according to the current locale.

4.18.2.2 char* setlocale (int *category*, const char * *locale*)

Select the part of the program's locale as specified by the category and locale parameters. The function can be used to change or query the program's current locale.

The default is supposed to be setlocale(LC_ALL, "C");

Parameters

category A macro to select the Locale portion to select.

See also

LC_ALL, etc...

Parameters

locale A string specifying the locale environment. Normally "C" for our use. If locale is NULL, setlocale will return the category of the current locale.

Returns

A pointer to a string associated with the category for the locale, or 0 on error.

4.19 include/propeller.h File Reference

Provides Propeller specific functions.

```
#include "cog.h"
#include <stdint.h>
#include <string.h>
```

Defines

- #define HUBDATA __attribute__((section(".hub")))

  HUBDATA tells compiler to put data into HUB RAM section. This is mostly useful in XMM modes where data must be shared in hub.

- #define HUBTEXT __attribute__((section(".hubtext")))

  HUBTEXT tells compiler to put code into HUB RAM section.

- #define PAR _PAR

  Parameter register is used for sharing HUB RAM address info with the COG.

- #define CNT _CNT

  The system clock count.

- #define INA _INA

  Use to read the pins when corresponding DIRA bits are 0.

- #define INB _INB

  Unused in P8X32A.

- #define OUTA _OUTA

  Use to set output pin states when corresponding DIRA bits are 1.

- #define OUTB _OUTB

  Unused in P8X32A.

- #define DIRA _DIRA

  Use to set pins to input (0) or output (1).

- #define DIRB _DIRB

  Unused in P8X32A.

- #define CTRA _CTRA

  Counter A control register.

- #define CTRB _CTRB

  Counter B control register.

- #define FRQA _FRQA

  Counter A frequency register.

- #define FRQB _FRQB

  Counter B frequency register.

- #define PHSA _PHSA

  Counter A phase accumulation register.

- #define PHSB _PHSB

  Counter B phase accumulation register.

- #define VCFG _VCFG

  Video Configuration register can be used for other special output.

- #define VSCL _VSCL

  Video Scale register for setting pixel and frame clocks.

- #define CLKFREQ _CLKFREQ

  Returns the current clock frequency.

- #define CLKMODE _CLKMODE

  Returns the current clock mode.

- #define clkset(mode, frequency)
  Set clock mode and frequency.
- #define cogid() __builtin_propeller_cogid()
  Return the id of the current cog.
- #define coginit(id, code, param)
  Start a cog with a parameter.
- #define cognew(code, param) coginit(0x8, (code), (param))
  Start a new Propeller PASM COG.
- #define cogstop(a) __builtin_propeller_cogstop((a))
  Stop a COG.
- #define locknew() __builtin_propeller_locknew()
  Get a new lock.
- #define lockret(lockid) __builtin_propeller_lockret((lockid))
  Return lock to pool.
- #define lockset(lockid) __builtin_propeller_lockset((lockid))
  Set a lock.
- #define lockclr(lockid) __builtin_propeller_lockclr((lockid))
  Clear lock.
- #define waitcnt(a) __builtin_propeller_waitcnt((a),0)
  Wait until system counter reaches a value.
- #define waitcnt2(a, b) __builtin_propeller_waitcnt((a),(b))
  Wait until system counter reaches a value.
- #define waitpeq(state, mask) __builtin_propeller_waitpeq((state), (mask))
  Wait until INA equal state & mask.
- #define waitpne(state, mask) __builtin_propeller_waitpne((state), (mask))
  Wait until INA not equal state & mask.
- #define waitvid(colors, pixels) __builtin_propeller_waitvid((colors), (pixels))
  Wait for video generator to accept pixel info.

Functions

- int cogstart (void(*func)(void *), void *par, void *stack, size_t stacksize)
  Start a new propeller LMM function/thread in another COG.

4.19.1 Detailed Description

Provides Propeller specific functions. Copyright (c) 2011-2012 by Parallax, Inc. MIT Licensed

Definition in file propeller.h.

4.19.2 Define Documentation

4.19.2.1 #define clkset(*mode, frequency*)

Value:

```
do { \
  _CLKFREQ = (frequency); \
  _CLKMODE = (mode); \
  __builtin_propeller_clkset(mode); \
} while(0)
```

Set clock mode and frequency.

This macro is used to set the run-time clock mode and frequency. The clock mode and frequency are normally configured by the loader based on the user selected board type.

Please see the Propeller Data Sheet for more clock information.

Parameters

mode The 8 bit clock mode

frequency The 32 bit clock frequency

Returns

This macro will not return a value.

Definition at line 89 of file propeller.h.

4.19.2.2 #define cogid( ) __builtin_propeller_cogid()

Return the id of the current cog.

Sometimes we need to know which COG is running the program. cogid returns that value.

Returns

ID number of current COG

Definition at line 102 of file propeller.h.

4.19.2.3 #define coginit(*id, code, param*)

Value:

```
__builtin_propeller_coginit( \
                                (((uint32_t)(param) << 16) & 0xfffc0000) \
                               |(((uint32_t)(code) <<  2) & 0x0003fff0) \
                               |(((id)                  ) & 0x0000000f) )
```

Start a cog with a parameter.

The fields in parameters are:

- 31:18 = 14-bit Long address for PAR Register
- 17:4 = 14-bit Long address of CODE to load
- 3 = New bit
- 2:0 = Cog ID. New bit 3 is 0.

It is important to realize that a 14 bit address means that long aligned addresses or pointers should be use. That is if you pass a value to the PAR such as 3, the value will be truncated to 0. A value 5 will be interpreted as 4.

Parameters

id The COG id to initialize

code Start address of PASM code to load.

param PAR address

Returns

COG ID provided by the builtin function or -1 on failure.

Definition at line 126 of file propeller.h.

4.19.2.4 #define cognew(*code, param*) coginit(0x8, (code), (param))

Start a new Propeller PASM COG.

This is use to start a COG with code compiled as PASM, AS, or COG-C. PASM can be any Spin/PASM code that is self-contained. That is, all data for initialization and mailbox use are passed via the par parameter. Changing PASM variables from SPIN code will not work with this method.

GAS and COG-C programs have similar restrictions. COG-C programs should not use any stack or variables in HUB memory that are not accessed via PAR mailbox or pointers.

Parameters

code Address of PASM to load

param Value of par parameter usually an address

Returns

COG ID provided by the builtin function or -1 on failure.

Definition at line 147 of file propeller.h.

4.19.2.5 #define cogstop(*a*) __builtin_propeller_cogstop((a))

Stop a COG.

Parameters

a The COG ID

Definition at line 153 of file propeller.h.

4.19.2.6 #define HUBTEXT __attribute__((section(".hubtext")))

HUBTEXT tells compiler to put code into HUB RAM section.

This is a GCC super-power. Put code in HUB RAM even in XMM modes. Sometimes code in XMM programs is time sensitive.

Use HUBTEXT before a function declaration to make sure code is run from HUB instead of external memory. Performance of code run from external memory is unpredictable across platforms.

Definition at line 36 of file propeller.h.

4.19.2.7 #define lockclr(*lockid*) __builtin_propeller_lockclr((lockid))

Clear lock.

Parameters

lockid

Definition at line 196 of file propeller.h.

4.19.2.8 #define locknew( ) __builtin_propeller_locknew()

Get a new lock.

Returns

new lockid

Definition at line 177 of file propeller.h.

4.19.2.9 #define lockret(*lockid*) __builtin_propeller_lockret((lockid))

Return lock to pool.

Parameters

lockid

Definition at line 183 of file propeller.h.

4.19.2.10 #define lockset(*lockid*) __builtin_propeller_lockset((lockid))

Set a lock.

Parameters

lockid

Returns

true on success

Definition at line 190 of file propeller.h.

4.19.2.11 #define waitcnt(*a*) __builtin_propeller_waitcnt((a),0)

Wait until system counter reaches a value.

Parameters

a Target value

Definition at line 202 of file propeller.h.

4.19.2.12 #define waitcnt2(*a, b*) __builtin_propeller_waitcnt((a),(b))

Wait until system counter reaches a value.

Parameters

a Target value

b Adjust value

Definition at line 209 of file propeller.h.

4.19.2.13 #define waitpeq(*state, mask*) __builtin_propeller_waitpeq((state), (mask))

Wait until INA equal state & mask.

Parameters

state Target value

mask Ignore masked 0 bits in state

Definition at line 216 of file propeller.h.

4.19.2.14 #define waitpne(*state, mask*) __builtin_propeller_waitpne((state), (mask))

Wait until INA not equal state & mask.

Parameters

state Target value

mask Ignore masked 0 bits in state

Definition at line 223 of file propeller.h.

4.19.2.15 #define waitvid(*colors, pixels*) __builtin_propeller_waitvid((colors), (pixels))

Wait for video generator to accept pixel info.

Parameters

colors A long containing four byte-sized color values, each describing the four possible colors of the pixel patterns in Pixels.

pixels The next 16-pixel by 2-bit (or 32-pixel by 1-bit) pixel pattern to display.

Definition at line 231 of file propeller.h.

4.19.3 Function Documentation

4.19.3.1 int cogstart (void(*)(void *) *func*, void * *par*, void * *stack*, size_t *stacksize*)

Start a new propeller LMM function/thread in another COG.

This function starts a new LMM VM kernel in a new COG with func as the start function. The stack size must be big enough to hold the struct _thread_state_t, the initial stack frame, and other stack frames used by called functions.

This function can be used instead of _start_cog_thread.

Parameters

func LMM start function

par Value of par parameter usually an address

stack Address of user defined stack space.

stacksize Size of user defined stack space.

Returns

COG ID allocated by the function or -1 on failure.

4.20 include/pthread.h File Reference

Provides API for implementation of pthread functions.

```
#include <sys/thread.h>
#include <sys/size_t.h>
#include <setjmp.h>
```

Classes

- struct pthread_attr_t

  The pthread_attr_t struct TODO.

- struct pthread_mutex_t

  The pthread_mutex_t TODO.

Defines

- #define PTHREAD_STACK_MIN 64

  Minimum stack size for a thread.

- #define _PTHREAD_STACK_DEFAULT 512

  Default stack size for a thread.

- #define _PTHREAD_DETACHED 0x0001

  Detached flag for the pthread "flags" field.

- #define _PTHREAD_TERMINATED 0x8000
  Terminated flag for the pthread "flags" field.

Typedefs

- typedef _pthread_state_t * pthread_t
  The pthread_t typedef is used as the "handle" for threads and functions.
- typedef struct pthread_attr_t pthread_attr_t
  The pthread_attr_t struct TODO.
- typedef struct pthread_mutex_t pthread_mutex_t
  The pthread_mutex_t TODO.
- typedef int pthread_mutexattr_t
  Mutex attributes are not implemented.

Variables

- atomic_t __pthreads_lock
  Lock for pthreads data structures.

4.20.1 Detailed Description

Provides API for implementation of pthread functions. POSIX threads are a set of functions that support applications with requirements for multiple flows of control, called threads, within a process. Multithreading is used to improve the performance of a program.

The pthread module provides a subset of the POSIX standard pthread library for allowing multiple threads to run on multiple COG instances of the PropellerGCC LMM interpreter. Multiple threads can also run on an XMM interpreter, but only one XMM interpreter can run at any given time.

Precondition

It is very important to understand that pthreads provide a cooperative non-preemptive multithreading system. The consequences are that certain rules will apply. I.E.

- The Propeller waitcnt() function can not be used.
- Delays can only be introduced with usleep() and its variations.
- Standard printf() and friends must be used.
- Do not use -Dprintf=__simple_printf with pthreads.
- Threads must not hog the LMM COG kernel; they must yield.
- Gobal data must be protected by mutex/semaphore.

Definition in file pthread.h.

Index